

E2 15-Puzzle Problem (IDA*)

16337102 ZiLin Huang

September 14, 2018

Contents

1	IDA* Algorithm	2
1.1	Description	2
1.2	Pseudocode	2
2	Tasks	3
3	Codes	3
4	Results	8

1 IDA* Algorithm

1.1 Description

Iterative deepening A* (IDA*) was first described by Richard Korf in 1985, which is a graph traversal and path search algorithm that can find the shortest path between a designated start node and any member of a set of goal nodes in a weighted graph.

It is a variant of **iterative deepening depth-first search** that borrows the idea to use a heuristic function to evaluate the remaining cost to get to the goal from the **A* search algorithm**.

Since it is a depth-first search algorithm, its memory usage is lower than in A*, but unlike ordinary iterative deepening search, it concentrates on exploring the most promising nodes and thus does not go to the same depth everywhere in the search tree.

Iterative-deepening-A* works as follows: at each iteration, perform a depth-first search, cutting off a branch when its total cost $f(n) = g(n) + h(n)$ exceeds a given threshold. This threshold starts at the estimate of the cost at the initial state, and increases for each iteration of the algorithm. At each iteration, the threshold used for the next iteration is the minimum cost of all values that exceeded the current threshold.

1.2 Pseudocode



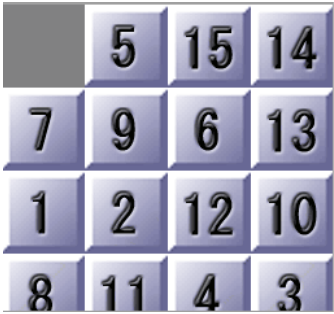

```
path          current search path (acts like a stack)
node          current node (last node in current path)
g            the cost to reach current node
f            estimated cost of the cheapest path (root..node..goal)
h(node)      estimated cost of the cheapest path (node..goal)
cost(node, succ) step cost function
is_goal(node) goal test
successors(node) node expanding function, expand nodes ordered by g + h(node)
ida_star(root) return either NOT_FOUND or a pair with the best path and its cost

procedure ida_star(root)
  bound := h(root)
  path := [root]
  loop
    t := search(path, 0, bound)
    if t = FOUND then return (path, bound)
    if t = ∞ then return NOT_FOUND
    bound := t
  end loop
end procedure

function search(path, g, bound)
  node := path.last
  f := g + h(node)
  if f > bound then return f
  if is_goal(node) then return FOUND
  min := ∞
  for succ in successors(node) do
    if succ not in path then
      path.push(succ)
      t := search(path, g + cost(node, succ), bound)
      if t = FOUND then return FOUND
      if t < min then min := t
      path.pop()
    end if
  end for
  return min
end function
```

2 Tasks

- Please solve 15-Puzzle problem by using IDA* (Python or C++). You can use one of the two commonly used heuristic functions: $h1$ = the number of misplaced tiles. $h2$ = the sum of the distances of the tiles from their goal positions.
- Here are 4 test cases for you to verify your algorithm correctness. You can also play this game (15puzzle.zip) for more information.

	<p>TextOut of Result</p> <pre> 11 3 1 7 4 6 8 2 15 9 10 13 14 12 5 0 LowerBound 36 moves A optimal solution 56 moves Used time 3 sec 13 10 8 6 9 12 5 13 10 8 12 15 14 5 13 12 15 14 5 13 14 9 4 11 3 1 6 4 11 3 1 6 4 2 8 10 12 15 10 8 7 4 2 11 3 5 9 10 11 3 6 2 3 7 8 12 </pre>		<p>TextOut of Result</p> <pre> 14 10 6 0 4 9 1 8 2 3 5 11 12 13 7 15 LowerBound 37 moves A optimal solution 49 moves Used time 0 sec 6 10 9 4 14 9 4 1 10 4 1 3 2 14 9 1 3 2 5 11 8 6 4 3 2 5 13 12 14 13 12 7 11 12 7 14 13 9 5 10 6 8 12 7 10 6 7 11 15 </pre>
	<p>TextOut of Result</p> <pre> 0 5 15 14 7 9 6 13 1 2 12 10 8 11 4 3 LowerBound 44 moves A optimal solution 62 moves Used time 4 sec 7 9 2 1 9 2 5 7 2 5 1 11 8 9 5 1 6 12 10 3 4 8 11 10 12 13 3 4 8 12 13 15 14 3 4 8 12 13 15 14 7 2 1 5 10 11 13 15 14 7 3 4 8 12 15 14 11 10 9 13 14 15 </pre>		<p>TextOut of Result</p> <pre> 6 10 3 15 14 8 7 11 5 1 0 2 13 12 9 4 LowerBound 32 moves A optimal solution 48 moves Used time 0 sec 9 12 13 5 1 9 7 11 2 4 12 13 9 7 11 2 15 3 2 15 4 11 15 8 14 1 5 9 13 15 7 14 10 6 1 5 9 13 14 10 6 2 3 4 8 7 11 12 </pre>

- Please send E02.YourNumber.pdf to ai_2018@foxmail.com, you can certainly use E02_15puzzle.tex as the L^AT_EX template.

3 Codes

```

#include<iostream>
#include<stack>
#include<list>
#include<stdlib.h>
#include<time.h>
using namespace std;

class node
{
public:
    node()
    {
        for(int i = 0;i < 4;i++)
            for(int n = 0;n < 4;n++)
                state[i][n] = i * 4 + n + 1;
        state[3][3] = 0;
        g = 0;
    }

```

```

        movement = 0;
    }
    node& operator=(const node& m)//
    {
        for(int i = 0;i < 4;i++)
            for(int n = 0;n < 4;n++)
                this->state[i][n] = m.state[i][n];
        this->g = m.g;
        this->movement = m.movement;
        return *this;
    }
    bool operator == (const node& m)
    {
        bool result = true;
        for(int i = 0;i < 4;i++)
            for(int n = 0;n < 4;n++)
                if(this->state[i][n] != m.state[i][n])
                    result = false;

        return result;
    }
    int state[4][4];//
    int g; //cost
    int movement;//
};

int h(const node it) //
{
    int num = 0;
    for(int i = 1;i < 16;i++)
    {
        int x = (i - 1) / 4;
        int y = (i + 3) % 4;
        for(int n = 0;n < 4;n++)
            for(int m = 0;m < 4;m++)
                if(it.state[n][m] == i)
                    num = num + abs(n - x) + abs(m - y);
    }
    return num;
}

class cmp
{
public:
    bool operator()(const node n1, const node n2)
    {
        return h(n1) < h(n2);
    }
};

```

```

bool is_goal(node m)
{
    bool result = true;
    for(int i = 0; i < 4; i++)
        for(int n = 0; n < 4; n++)
            if(m.state[i][n] != i*4 + n + 1 && i*4 + n + 1 != 16)
                result = false;
    return result;
}

bool in_Path(node &n, list<node> &path)
{
    bool result = false;
    list<node>::iterator i;
    for(i = path.begin(); i != path.end(); i++)
        if(n == *i)
        {
            result = true;
            break;
        }
    return result;
}

list<node> successor(node &n)
{
    list<node> result;
    int i, m;
    int xx, yy;
    for(i = 0; i < 4; i++)//
        for(m = 0; m < 4; m++)
            if(n.state[i][m] == 0)
            {
                xx = i;
                yy = m;
            }
    if(xx + 1 < 4)//
    {
        node x = n;
        x.movement = x.state[xx+1][yy];
        x.state[xx][yy] = x.state[xx+1][yy];
        x.state[xx+1][yy] = 0;
        x.g++;
        result.push_back(x);
    }
    if(xx - 1 >= 0)
    {
        node x = n;
        x.movement = x.state[xx-1][yy];
        x.state[xx][yy] = x.state[xx-1][yy];

```

```

        x.state[xx-1][yy] = 0;
        x.g++;
        result.push_back(x);
    }
    if(yy + 1 < 4)
    {
        node x = n;
        x.movement = x.state[xx][yy + 1];
        x.state[xx][yy] = x.state[xx][yy + 1];
        x.state[xx][yy + 1] = 0;
        x.g++;
        result.push_back(x);
    }
    if(yy - 1 >= 0)
    {
        node x = n;
        x.movement = x.state[xx][yy-1];
        x.state[xx][yy] = x.state[xx][yy-1];
        x.state[xx][yy-1] = 0;
        x.g++;
        result.push_back(x);
    }
    result.sort(cmp()); //
    return result;
}

int search_path(list<node> &path, int g, int bound)
{
    node n = path.back();
    int f = g + h(n);
    if(f > bound)
        return f;
    if(is_goal(n))
        return -1; //found
    int mini = -2; //
    list<node> successors = successor(n);
    list<node>::iterator it;
    for(it = successors.begin(); it != successors.end(); it++)
    {
        if(!in_Path(*it, path))
        {
            path.push_back(*it);
            //cout << (*it).movement << endl;
            int t;
            t = search_path(path, g + 1, bound);
            if(t == -1) //found
                return -1;
            if(t < mini || mini < 0)
                mini = t;
        }
    }
}

```

```

        path.pop_back();
    }
}
return mini;
}

int main()
{
    time_t start, stop;
    start = time(NULL);
    node root;

    root.state[0][0] = 5;
    root.state[0][1] = 1;
    root.state[0][2] = 7;
    root.state[0][3] = 3;

    root.state[1][0] = 2;
    root.state[1][1] = 0;
    root.state[1][2] = 6;
    root.state[1][3] = 4;

    root.state[2][0] = 9;
    root.state[2][1] = 10;
    root.state[2][2] = 11;
    root.state[2][3] = 8;

    root.state[3][0] = 13;
    root.state[3][1] = 14;
    root.state[3][2] = 15;
    root.state[3][3] = 12;

    for(int i = 0; i < 4; i++)
    {
        for(int n = 0; n < 4; n++)
            cout << root.state[i][n] << ' ';
        cout << endl;
    }

    int bound = h(root);
    list<node> path;
    path.push_back(root);
    int t;
    while(true)
    {
        t = search_path(path, 0, bound);
        if(t == -1)
            break;
        if(t == -2)

```

```

    {
        cout << "not_found" << endl;
        break;
    }
    bound = t;
}
stop = time(NULL);
list<node>::iterator it;
cout << "LowerBound_" << bound << "_moves" << endl;
cout << "A_optimal_solution_" << path.size() - 1 << "_moves" << endl;
cout << "Used_time_" << stop - start << "_sec" << endl;
for(it = path.begin(); it != path.end(); it++)
    if((*it).movement != 0)
        cout << (*it).movement << '_';
cout << endl;
return 0;
}

```

4 Results

```

5 1 7 3
2 0 6 4
9 10 11 8
13 14 15 12
LowerBound 10 moves
A optimal solution 10 moves
Used time 0 sec
2 5 1 2 6 7 3 4 8 12

Process returned 0 (0x0)    execution time : 0.380 s
Press any key to continue.

```